МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Линейные структуры данных: стек, очередь, дек
Вариант 11 е-в

Студент гр. 8304 _____ Николаева М. А.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2019

Цель работы.

Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач.

Постановка задачи.

Рассматривается выражение следующего вида:

< выражение >::= < терм > | < терм > + < выражение > |
< терм > – < выражение >
< терм > ::= < множитель > | < множитель > * < терм >
< множитель > ::= < число > | < переменная > | (< выражение >) |
< множитель > ^ < число >
< число > ::= < цифра >
< переменная > ::= < буква >

Такая форма записи выражения называется ***инфиксной***.

Постфиксной (префиксной) формой записи выражения *aDb* называется запись, в которой знак операции размещен за (перед) операндами: *abD* (*Dab*).

Примеры

Инфиксная	***Постфиксная***	***Префиксная***
a–b	*ab–*	*–ab*
*a*b+c*	*ab*c+*	*+*abc*
a(b+c)*	*abc+**	**a+bc*
*a+b^c^d*e*	*abc^d^e*+*	*+a*^b^cde*.

Отметим, что постфиксная и префиксная формы записи выражений не содержат скобок.

Вариант 11 е-в: вывести в обычной (инфиксной) форме выражение, записанное в префиксной форме в заданном текстовом файле prefix (рекурсивные процедуры не использовать и лишние скобки не выводить);

Описание алгоритма.

Изначально программа должна считать данные и передать строку в функцию преобразования выражения из префиксной формы в инфиксную. Далее для перевода выражения из префиксной в инфиксную запись необходимо следовать алгоритму:

1) Проходим по строке справа налево.
2) Если читаем число, то заносим его в стек
3) Если читаем знак операции, то:
   а) Берем текущий знак операции и следующий

   б) Если в первом элементе приоритет операции меньше (и не равен 0), чем у рассматриваем операции, то берем первый элемент в скобки

   в) Аналогично для 2-го элемента

   г) Записываем в стек строку вида: 2-й элемент + знак операции + 1-й элемент

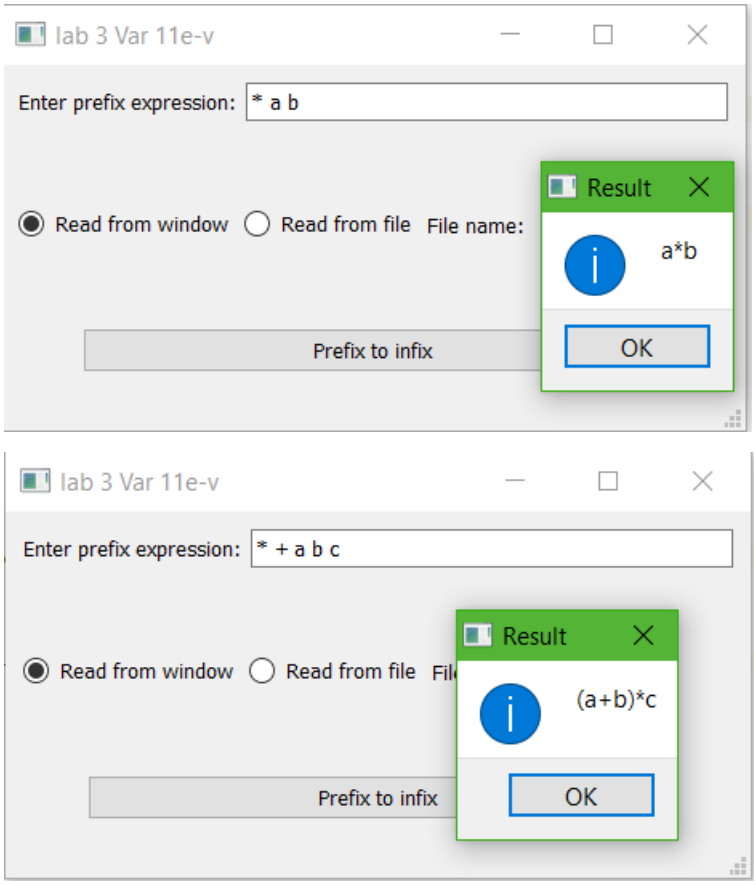
4) Если строка полностью пройдена, то результатом является значение вершины стека

Спецификация программы.

Программа предназначена для перевода записи выражения из префиксной в инфиксную.

Программа написана на языке C++ с использованием фреймворка Qt. Входными данными являются цифры и знаки арифметических операций, считываемые из файла prefix.txt, дополнительно реализована возможность ввода с помощью GUI. Выходными данными являются промежуточные значения и конечный результат. Данные выводятся в QMessageBox.

Рисунок 1- Результат работы программы



Тестирование.

Таблица 1 – Результаты тестирования программы

Input	**Output**
+ a b	a+b
* + a b c	(a+b)*c
^ - * a b + c d 11	(a*b-c+d)^11
123	123
- 100 + * h j ^ h j	100-h*j+h^j
+ a b b	Введенное выражение некорректно
a c	Введенное выражение некорректно
^ - * a b + c d4 11	Введенное выражение некорректно
-	Введенное выражение некорректно

- 10d0 + * h j ^ h j	Введенное выражение некорректно
12 12 13	Введенное выражение некорректно
1 1 1 +	Введенное выражение некорректно

Анализ алгоритма.

Алгоритм работает за линейное время от размера строки. Для экономии памяти строка передается по константной ссылке.

Описание функций и СД.

Класс Stack реализует структуру стека, а также методы для работы с ним.

Стандартные методы для работы со стеком:

```
void push(const Data elem);
void pop();
Data top() const;
size_t size() const;
bool isEmpty() const;
```

Класс Mainwindow реализует алгоритм перевода выражения из префиксной формы в инфиксную с помощью стека.

Статический метод для перевода выражения из постфиксной формы в инфиксную:

```
static std::string prefixToInfix(const std::string& expression);
```

Принимает на вход константную ссылку на строку-выражение, возвращает выражение в инфиксной форме, если исходное выражение корректно и пустую строку в случае ошибки. Строка анализируется посимвольно справа налево. Если текущий символ "число или буква", тогда элемент помещается в стек с приоритетом 0, если текущий символ "знак операции", из стека достаются два элемента, записываются в временную строку с учетом приоритетов операций и временная строка помещается в стек с приоритетом знака. В ходе преобразования, если стек пустой, выводится ошибка и возвращается пустая строка. После

преобразования в стеке должен находиться один элемент - инфиксное выражение.

Выводы.

В ходе работы были приобретены навыки работы со стеком, изучены методы работы с ним (объявлять, заносить в него переменных и забирать их). Был изучен и реализован алгоритм перевода записи из префиксной в инфиксную.

Приложение А. Исходный код программы.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#define QT_NO_DEBUG_OUTPUT

#include <QMainWindow>
#include <QDebug>
#include <string>
#include <QString>
#include <QMessageBox>
#include <QTextStream>
#include <QFile>
#include <QFileDialog>
#include "stack.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_prefixToInfix_pushButton_clicked();
    void on_readFromFile_radioButton_clicked();
    void on_readFromWindow_radioButton_clicked();

public slots:
    static std::string prefixToInfix(const std::string& expression);
    static bool isSign(const char ch);
    static bool isAlpha(const char ch);
    static bool isDigit(const char ch);

private:
    Ui::MainWindow *ui;
    QTextStream* in;
    QFile* file;
};

#endif // MAINWINDOW_H
```

stack.h

```
#ifndef STACK_H
#define STACK_H

#include <QObject>
#include <vector>
#include <QDebug>
#include <string>


enum class optype {power = 3, multiply = 2, minus = 1, plus = 1, null = 0};


typedef std::pair<std::string, optype> Data;


class Stack : public QObject
{
    Q_OBJECT
public:
    explicit Stack(QObject *parent = nullptr);
    ~Stack() = default;

    Stack(const Stack&) = delete;
    Stack& operator=(const Stack&) = delete;
```

```
public slots:
    void push(const Data elem);
    void pop();
    Data top() const;
    size_t size() const;
    bool isEmpty() const;

private:
    std::vector <Data> stack;
    std::size_t sizeStack;
};

#endif // STACK_H
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    w.setWindowTitle("lab 3 Var 11e-v");
    return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    in = new QTextStream;
    file = new QFile;

    ui->setupUi(this);
    ui->readFromWindow_radioButton->setChecked(true);
}


MainWindow::~MainWindow()
{
    delete file;
    delete in;
    delete ui;
}


void MainWindow::on_readFromFile_radioButton_clicked()
{
    file->close();

    ui->textLabel_0->setEnabled(false);
    ui->input_lineEdit->setEnabled(false);
    ui->input_lineEdit->clear();

    QString fileName = QFileDialog::getOpenFileName(this, "Open file", "", "*.txt");
    file->setFileName(fileName);
    file->open(QFile::ReadOnly | QFile::Text);
    in->setDevice(file);

    if (fileName != "")
        ui->fileName_textLabel->setText(fileName);
    else
        ui->fileName_textLabel->setText("none");
}


void MainWindow::on_readFromWindow_radioButton_clicked()
{
```

```
    file->close();

    ui->textLabel_0->setEnabled(true);
    ui->input_lineEdit->setEnabled(true);
    ui->input_lineEdit->clear();
    ui->fileName_textLabel->setText("none");
}


void MainWindow::on_prefixToInfix_pushButton_clicked()
{
    std::string tmpStr;

    if (ui->readFromWindow_radioButton->isChecked()) {
        tmpStr= ui->input_lineEdit->text().toStdString();
    }
    else {
        tmpStr = in->readLine().toStdString();
    }

    std::string result = MainWindow::prefixToInfix(tmpStr);

    if (result != "") {
        qDebug() << "Выражение в инфиксной форме: " << result.c_str();
        QMessageBox::information(this, "Result", result.c_str());
    }
    else {
        qDebug() << "Введенное выражение некорректно";
        QMessageBox::warning(this, "Result", "Введенное выражение некорректно");
    }

    ui->input_lineEdit->clear();
    qDebug() << "End";
}


bool MainWindow::isDigit(const char ch)
{
    return (ch >= '0' && ch <= '9');
}


bool MainWindow::isAlpha(const char ch)
{
    return ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'));

}


bool MainWindow::isSign(const char ch)
{
    return (ch == '+' || ch == '-' || ch == '*' || ch == '^');
}


std::string MainWindow::prefixToInfix(const std::string &expression)
{
    Stack stack;
    std::string tmpStr;
    for (auto i = expression.crbegin(); i < expression.crend(); ++i) {
        char elem = *i;
        if (elem == ' ') {
            continue;
        }
        else if (isDigit(elem) || isAlpha(elem) ) {
            bool isDigitElem = isDigit(elem);
            while (elem != ' ' && i != expression.crend()) {
                if ((isDigitElem && isAlpha(elem)) ||
                        (!isDigitElem && !isAlpha(elem))) {
                    qDebug() << "Error: wrong data!" << elem;
                    return "";
                }
                tmpStr += elem;
                ++i;
                elem = *i;

            }
            std::reverse(tmpStr.begin(), tmpStr.end());
            stack.push(Data(tmpStr, optype::null));
```

```
            qDebug() << "В стек помещен элемент:" << tmpStr.c_str();
        }
        else if (isSign(elem)) {
            Data firstArg;
            Data secondArg;
            qDebug() << "Знак оперции:" << elem;
            if (!stack.isEmpty()) {
                secondArg = stack.top();
                stack.pop();
            }
            else {
                qDebug() << "Error: stack is empty";
                return "";
            }
            if (!stack.isEmpty()) {
                firstArg = stack.top();
                stack.pop();
            }
            else {
                qDebug() << "Error: stack is empty!";
                return "";
            }
            qDebug() << "Выражения в стеке: " << firstArg.first.c_str() <<
                        "и" << secondArg.first.c_str();
            optype tmpOptype = optype::null;
            if (elem == '+') {
                tmpOptype = optype::plus;
            } else if (elem == '-') {
                tmpOptype = optype::minus;
            } else if (elem == '*') {
                tmpOptype = optype::multiply;
            } else if (elem == '^') {
                tmpOptype = optype::power;

            }

            if (secondArg.second != optype::null && secondArg.second < tmpOptype) {
                tmpStr += '(';
                tmpStr += secondArg.first;
                tmpStr += ')';
            } else {
                tmpStr += secondArg.first;
            }

            tmpStr += elem;

            if (firstArg.second != optype::null && firstArg.second < tmpOptype) {
                tmpStr += '(';
                tmpStr += firstArg.first;
                tmpStr += ')';
            } else {
                tmpStr += firstArg.first;
            }

            qDebug() << "Итоговое выражение помещается в стек:" << tmpStr.c_str();
            stack.push(Data(tmpStr, tmpOptype));
        }
        else {
            qDebug() << "Некорректный символ в строке!";
            return "";
        }
        tmpStr = "";
    }

    if (stack.size() == 1) {
        tmpStr = stack.top().first;
        stack.pop();
        return tmpStr;
    }
    else {
        qDebug() << "Error: string is incorrect!" << stack.size() ;
        return "";
    }
}
```

stack.cpp

```
#include "stack.h"

Stack::Stack(QObject *parent) : QObject(parent)
{
    sizeStack = 0;
}


size_t Stack::size() const
{
    return sizeStack;
}


bool Stack::isEmpty() const
{
    return size() == 0;
}


Data Stack::top() const
{
    if (!isEmpty()) {
        return stack.back();
    }
    else {
        qDebug() << "Error. Stack is empty";
        return Data("", optype::null);
    }
}


void Stack::pop()
{
    if (!isEmpty()){
        stack.pop_back();
        sizeStack -= 1;
    }
    else {
        qDebug() << "Error. Stack is empty";
    }
}


void Stack::push(const Data elem)
{
    stack.push_back(elem);
    sizeStack += 1;
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="enabled">
   <bool>true</bool>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>465</width>
    <height>228</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
     <layout class="QHBoxLayout" name="horizontalLayout">
```

```
  <item>
   <widget class="QLabel" name="textLabel_0">
    <property name="text">
     <string>Enter prefix expression:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLineEdit" name="input_lineEdit"/>
  </item>
 </layout>
</item>
<item row="1" column="0">
 <layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
   <widget class="QRadioButton" name="readFromWindow_radioButton">
    <property name="text">
     <string>Read from window</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QRadioButton" name="readFromFile_radioButton">
    <property name="text">
     <string>Read from file</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="textLabel_1">
    <property name="text">
     <string>File name:</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QLabel" name="fileName_textLabel">
    <property name="text">
     <string>none</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item row="2" column="0">
 <layout class="QHBoxLayout" name="horizontalLayout_3">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Preferred</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="prefixToInfix_pushButton">
    <property name="text">
     <string>Prefix to infix</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeType">
     <enum>QSizePolicy::Preferred</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
```

```
          <width>40</width>
          <height>20</height>
         </size>
        </property>
       </spacer>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>465</width>
     <height>26</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```